NISTIR 6294

Initial CAALS Device Capability Dataset

Version 1.0.7

Torsten A. Staab

Gary W. Kramer

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Chemical Science and Technology Laboratory
Analytical Chemistry Division
Gaithersburg, MD 20899

NIST

NISTIR 6294

Initial CAALS Device Capability Dataset

Version 1.0.7

Torsten A. Staab

Gary W. Kramer

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Chemical Science and Technology Laboratory
Analytical Chemistry Division
Gaithersburg, MD 20899

December 1998

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Gary R. Bachula, Acting Undersecretary for Technology

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Raymond G. Kammer, Director

Disclaimer:
This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor the CAALS members, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, product, or process disclosed, or represents that its use would not infringe upon privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government and shall not be used for advertising or product endorsement purposes.

The participation of individuals in the development of this document shall not be construed as an unreserved endorsement of the content of the document by the individuals or their employers.

Acknowledgments:
NIST would like to acknowledge the following individuals who have made substantial contributions to the concept and content of this document:
Uwe Bernhöft, National Institute of Standards and Technology
Martin J. Burns, Hypertek, Inc.
James T. Currie, Jr., Currie and Associates
Anthony J. Day
James R. DeVoe, National Institute of Standards and Technology (ret.)
John W. Elling, Los Alamos National Laboratory
Lawrence G. Falkenau, E.I Dupont de Nemours, Inc.
Peter J. Grandsard, Amgen, Inc.
J. Michael Griesmeyer, Sandia National Laboratories
Franklin R. Guenther, National Institute of Standards and Technology
David R. Hawk, Hewlett-Packard Co., Inc.
Gary W. Kramer, National Institute of Standards and Technology
Richard S. Lysakowski, Team Science, Inc.
James V. Petersen, Amgen, Inc.
Mark F. Russo, Bristol-Myers Squibb Co.
Marc L. Salit, National Institute of Standards and Technology
Reinhold Schäfer, Fachhochschule Wiesbaden
Brian D. Seiler, Eastman Chemical Co., Inc.
William J. Sonnefeld, Eastman Kodak Co., Inc.
Torsten A. Staab, National Institute of Standards and Technology
John Upham, ABC Laboratories, Inc.

This project is funded in part by NIST's Systems Integration for Manufacturing Applications (SIMA) Program. Initiated in 1994 under the federal government's High Performance Computing and Communications effort, SIMA is addressing manufacturing systems integration problems through applications of information technologies and development of standards-based solutions. With technical activities in all of NIST's laboratories covering a broad spectrum of engineering and manufacturing domains, SIMA is making information interpretable among systems and people within and across networked enterprises.

Availability:
Paper copies of this document are available from the National Technical Information Service. This document is available electronically on the Internet from the CAALS ftp site (caals.nist.gov) in Microsoft Word rich text format (RTF) and as a Postscript document. The document may be downloaded in ASCII mode, and the RTF-formatted files may be opened with retention of figures and formatting using either Microsoft Word for Macintosh 6.0.1 or Word for Windows 6.0 or higher.

Corrections:
Error reports, corrections, and comments concerning this document are welcomed and should be directed to Gary W. Kramer, NIST, 100 Bureau Drive, Stop 8394, Gaithersburg, MD 20899-8394; email: gary.kramer@nist.gov; telephone: (301) 975-4132; telefax: (301) 977-0587.

Table of Contents

1. Overview

1.1 Abstract

A Device Capability Dataset (DCD) describes the idiosyncratic characteristics of laboratory equipment, such as the equipment's identity, physical dimensions, location, supported command set, generated events, input-output (I/O) ports, and other resources. The DCD concept provides a means for standardizing the interfacing of laboratory automation devices in a descriptive rather than a prescriptive manner. Within the CAALS (Consortium on Automated Analytical Laboratory Systems) modular architecture logical model [1] this information is used by the Task Sequence Controller (TSC) to determine which tasks a Standard Laboratory Module (SLM) can perform.

1.2 Introduction

The Consortium on Automated Analytical Laboratory Systems (CAALS) was managed by the National Institute of Standards and Technology (NIST) from 1990 until 1996 as a part of NIST's mandates to help U.S. industry improve its competitiveness in the world marketplace and to develop standards for commerce. NIST, in conjunction with leaders from the private sector and the U.S. Department of Energy (DoE), formed CAALS to foster the development of automation for analytical chemistry. Ultimately, over 25 companies and agencies participated in the Consortium [2].

CAALS developed and promoted the concept of analytical equipment as interconnectable modules. Such modules can be easily linked to create measurement systems tailored to particular analyses. In a CAALS system, analytical instruments are intelligent modules in a hierarchical scheme—slave devices operating under the direction of workcell controllers [1].

There are two approaches to interfacing standards: prescriptive and descriptive. The former approach seeks to make all devices look and behave in the same fashion to the controller. The latter concept is to employ a common way to describe the idiosyncrasies of the device and its behavior. Where aspects of devices can easily be made similar, such as communication hardware and software, the prescriptive method is a powerful solution. The 9-pin EIA-232 serial and IEEE-1284 bi-directional parallel ports found on modern PCs and interfacing standards such as GPIB (IEEE-488) and the TCP/IP components of network connections are good examples of prescriptive standards. But such standards require the development of consensus in a market sphere that dwarfs the world of laboratory automation. Prescriptive standards can also be constraining to future progress, because new concepts have to be exceptionally promising to gain acceptance over entrenched standards. Finally, some instrument manufacturers, believing that prescriptive standards limit their ability to differentiate their products from those of their competitors, are apprehensive about supporting prescriptive interfacing standards.

CAALS research on simplifying laboratory system integration determined that there were areas where prescriptive, rule-based standards could be employed to advantage. Inter-device communication is a good example where existing standards coupled with new CAALS-generated concepts and guidelines can be used to solve many of the difficulties in incorporating equipment into an analytical system. The CAALS-I Communication Specification [3] provides a low-level protocol for controller-to-device communications. The CAALS High-Level

Communications Protocol [4] and the CAALS Common Command Set [5] complete the communications suite. Other problems could be overcome by ensuring that devices to be incorporated in the system exhibit appropriate “automation friendly” behavior [6,7]. However, the rich mix of instrumentation and methodologies employed in analytical chemistry coupled with the constant innovation in this field precludes a total solution to system integration standards based solely on rules. The CAALS research concluded that an additional automation concept was needed to handle the idiosyncrasies of individual instruments. Such a concept would allow for different ways of doing things, but the device’s functions and parameters would be described and expressed in a standard manner to preclude the need to generate a custom program to handle each variation. Thus the concept of the device capability dataset was born.

Once “novel” concepts are envisioned, parallel ideas are often discovered. The CAALS device capability dataset scheme is similar to the notion of PostScript Printer Description files (PPD files) conceived by Adobe Systems, Inc. (Mountain View, CA) to allow generic printer driver software to access and utilize the special features of a wide variety of PostScript rendering devices. Adobe Systems developed the concept to provide a uniform approach to using the diverse special features found in devices that contain PostScript interpreters. PPD files are human-readable, machine-parsable text files that allow generic printer drivers to support a variety of printer features such as different page sizes, color or black and white printing, simplex or duplex printing, film handling, varying font availability, document sorting, and stapling. “The information contained in PPD files serves as a list of available features, as a basis for building a user interface, and as a mechanism for invoking the features on a particular device” [8]. The DCD concept was conceived independently of the PPD notion and extends the descriptive standard idea to a much larger class of devices and scope of control.

This document defines the structure for a Capability Dataset of a Standard Laboratory Module (SLM) [1]. The SLM Capability Dataset is specified in the EXPRESS language [9,10], which is part of the ISO 10303 International Standard. ISO 10303 is commonly known as STEP (Standard for the Exchange of Product Model Data) [11].

This initial DCD specification is divided into three chapters and two appendices. Chapter One of the document gives an introduction to DCDs and STEP. The second chapter talks about the concept of DCDs. In the third chapter, the schema of a DCD is discussed. Appendix A lists the basic data types used to describe the DCD in EXPRESS language notation. Appendix B discusses DCDs in clear text format and includes an example of a DCD file. This report describes the *initial* CAALS DCD. The DCD categories and their represented data and data types are still under development, and therefore, those described here are neither final nor complete.

2. Concept

2.1 Static versus Dynamic Characteristics

Not all the data in DCD categories have the same temporal nature. Some information such as the original device manufacturer's name will not change throughout the lifetime of the device and can be considered static. Other facts, such as resource amounts and consumables, change during the operation of the device and must be accounted for dynamically. There are cases between these two extremes that range from semi-static to semi-dynamic. Analytical method names could be an example of semi-static data, while a device's initial fill level might constitute semi-dynamic data. The exact temporal attributes assigned are less important than the concept that data in a DCD can vary in both their time of inclusion and their longevity. A multi-author, multi-session DCD creation process is implicit in this notion. A DCD is created along a multi-step path beginning with the instrument manufacturer who imparts much of the static data. The system integrator and methods developer supply additional static, semi-static, and semi-dynamic data. And the system itself, during run time, updates many of the dynamic data.

2.2 Storage

There are many ways to store and maintain DCDs. For instance, an SLM may store its capability dataset in a plain ASCII file or in a database on its local hard drive. The static parts of the DCD information could be stored in the SLM's ROM, on disk, or in a remote database provided by the SLM manufacturer. The dynamic part could reside in an EPROM, Flash-ROM, floppy disc, or in memory.

In the CAALS modular architecture model, the Task Sequence Controller (TSC) stores the DCD of each connected SLM in a database, called the device database (Device DB) [1].

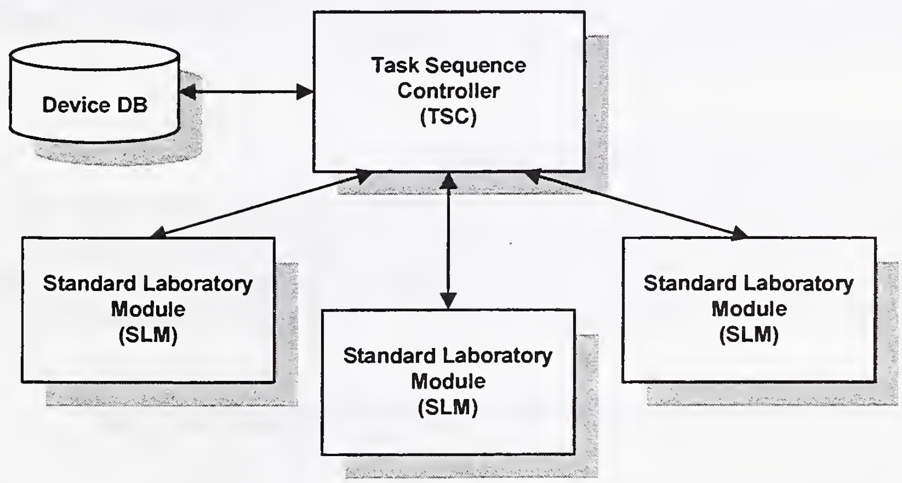

Figure 1. A portion of the CAALS modular architecture logical model.

CAALS does not impose any limitations on the mechanism used to store DCDs. However, to foster the platform- and device-independent structure and exchange of DCDs, CAALS uses the modeling language EXPRESS [9,10], a part of the ISO 10303 Standard [11], to represent the DCDs in system-independent form.

> ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular computer system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases, and archiving. ISO 10303 permits different implementation technologies to be used for storing, accessing, transferring, and archiving product data. The description of product data in integrated resource and application protocols requires the use of a format data specification language to ensure consistency and avoid ambiguity. The language is intended to be both human-readable and computer-interpretable to facilitate human understanding and to facilitate the generation of computer-interpretable application and supporting tools. [11]

2.3 Access

It is essential that the Task Sequence Controller (TSC) have access to the DCD for each SLM under its control. Such data may be uploaded directly from the SLM to the TSC. The static portion of a DCD could also be imported into the TSC's device database from an SLM driver disk or perhaps from a remote database maintained by the SLM manufacturer. In this version of the DCD specification, the system access points for the DCDs are not yet defined. However, NIST may specify these access points in more detail in a future release.

2.4 Updating

There are basically two ways for the TSC to keep track of the dynamic components of the Device Capability Dataset. One way is for the SLM to notify the TSC whenever its characteristics change. Another possibility is for the TSC to poll the SLM for an update on the characteristics. A combination of these techniques may be applied as well.

2.5 SLM Geometry and Topology

To describe the physical dimensions and locations of SLMs, ports, and resources, different geometric and topological models can be applied. The geometric and topological models used in this Device Capability Dataset specification are defined by the ISO 10303-42 standard [12]. ISO 10303-42 allows a standardized representation and exchange of geometric and topological data.

Part 42 of the ISO 10303 standard defines the following geometric models:

- SOLID MODEL
  A solid model is a complete representation of the nominal shape of a product such that all points in the interior are connected. Any point can be classified as being inside, outside or on the boundary of a solid.

- SURFACE MODEL
  Some product model representations consist of collections of surfaces that do not necessarily form the complete boundary of a solid. Such a model can be represented by a collection of faces.

- WIREFRAME MODEL
  A wireframe representation of a mechanical piece/part contains information only about the intersections of the surfaces forming the boundary of the part but does not contain information about the surfaces themselves.

- GEOMETRIC SET
  This model is intended for the transfer of models when a topological structure is not available.

3. Schema of Device Capability Dataset

The data that describe the characteristics of a Standard Laboratory Module (SLM) can be logically grouped into types [13]. This chapter describes these characteristic groups. In addition to, a textual description of each group, there is also a representation given in EXPRESS syntax [9,10]. The EXPRESS data types and entities that are used, but which are not defined in detail in this chapter, are listed in Appendix A.

3.1 Administrative

The ADMINISTRATIVE type provides the data for the identification of an SLM. The identification includes information such as the name of the SLM manufacturer, the model number, the serial number, the version number of the SLM software, the DCD version number, address information about software updates, support, technical information, the version number of the supported CAALS low-level protocol, and the size of the command buffer.

MANUFACTURER	name of the SLM manufacturer
MODEL NUMBER	model number of the SLM
SERIAL NUMBER	serial number of the SLM
SOFTWARE VERSION NUMBER	version number of the SLM's application or operating system software
DCD VERSION NUMBER	version number of the DCD instance; to be incremented on each revision
UPDATE ADDRESS	Internet URL (Universal Resource Locator) for software updates
SUPPORT ADDRESS	Internet URL for technical support
INFORMATION ADDRESS	Internet URL for technical information
COMMAND BUFFER SIZE	number of commands that the SLM can buffer (>=0; 0 means no buffering → immediate processing of received command)

EXPRESS Syntax:

```
ENTITY administrative;
  manufacturer            : STRING;
  model_number            : STRING;
  serial_number           : STRING;
  software_version_number : OPTIONAL STRING;
  dcd_version_number      : STRING;
  update_address          : OPTIONAL STRING;
  support_address         : OPTIONAL STRING;
  information_address     : OPTIONAL STRING;
  command_buffer_size     : buffer_size;
UNIQUE
  manufacturer, model_number, serial_number;
END_ENTITY;
```

3.2 Alarm

The ALARM type is used to describe alarms that may occur during operation of the SLM. Each alarm entry is specified by an identifier, an alarm description, a category, a priority level, a description of the alarm data format (syntax), and the possible recovery directions.

ALARM ID	alarm identifier
DESCRIPTION	description of alarm
PRIORITY	priority of the alarm; represents the severity of the alarm
CATEGORY	alarm category
ALARM DATA FORMAT	format of data that are returned with this alarm
RECOVERY COMMANDS	list of commands to recover from alarm situation

EXPRESS Syntax:

```
ENTITY alarm;
  alarm_id          : identifier;
  description       : text;
  priority          : priority_level;
  category          : OPTIONAL label;
  alarm_data_format : OPTIONAL SET [1:?] OF argument;
  recovery_commands : OPTIONAL SET [1:?] OF command_reference;
UNIQUE
  alarm_id;
END_ENTITY;
```

3.3 Command

The COMMAND type is used to describe commands supported by the SLM. Each command entry provides the information the TSC needs to know to invoke this command. It includes information on how to call the operation, which and how many resources are needed, the duration, and any necessary configurations.

COMMAND ID	command identifier
DESCRIPTION	description of the operation
CATEGORY	command category
FORMAL ARGUMENTS	list of formal arguments for this command
DURATION	required time to perform the command
MEASUREMENT BOUNDS	measurements bounds within the SLM can conduct the command
REQUIRED RESOURCES	list of resources needed to run the command
REQUIRED CONFIGURATIONS	list of configurations needed to run the command
PRODUCED RESOURCES	list of resources that will be produced during the run of the command
PORT INPUTS	information about required port input for this command, e.g., data or materials
PORT OUTPUTS	information about generated port output, e.g., data or materials
RESPONSE DATA	description of response data that is returned to the TSC during or upon completing the command

EXPRESS Syntax:

```
ENTITY command;
  command_id                 : identifier;
  description                : text;
  category                   : label;
  formal_arguments           : OPTIONAL SET [1:?] OF argument;
  duration                   : OPTIONAL coordinated_local_time;
  measurement_bounds         : OPTIONAL SET [1:?] OF range;
  required_resources         : OPTIONAL SET [1:?] OF
                               required_resource_reference;
  required_configurations    : OPTIONAL SET [1:?] OF
                               configuration_reference;
  produced_resources         : OPTIONAL SET [1:?] OF resource_reference;
  port_inputs                : OPTIONAL SET [1:?] OF port_io;
  port_outputs               : OPTIONAL SET [1:?] OF port_io;
  response_data              : OPTIONAL SET [1:?] OF data;

UNIQUE
  command_id;
END_ENTITY;
```

3.4 Event

The EVENT type is used to describe events that the SLM may generate and report during operation. Each event entry is specified by an identifier, a description, a priority level, a category, and a description of the event data format (syntax).

EVENT ID	event identifier
DESCRIPTION	description of event
PRIORITY	priority of the event
CATEGORY	event category
EVENT DATA FORMAT	format of data that is returned with this event

EXPRESS Syntax:

```
ENTITY event;
  event_id          : identifier;
  description       : text;
  priority          : priority_level;
  category          : label;
  event_data_syntax : OPTIONAL SET [1:?] OF argument;
UNIQUE
  event_id;
END_ENTITY;
```

3.5 Interaction

The INTERACTION type is used to describe an interaction between the TSC and the SLM. An interaction is a sequence of request and reply messages. Each interaction can be described by a unique interaction identifier, an optional description of the interaction, the type of interaction, and a list of transactions that constitute the interaction.

INTERACTION ID	interaction identifier
DESCRIPTION	description of interaction
TYPE	type of interaction (required or optional, and primary or secondary)
TRANSACTIONS	set of messages exchanged between controller and SLM

EXPRESS Syntax:

```
ENTITY interaction
  interaction_id : identifier;
  description    : text;
  type           : interaction_type;
  transactions   : SET [1..?] OF request_reply;
```

```
UNIQUE
  interaction_id;
END_ENTITY;
```

3.6 Maintenance

The MAINTENANCE type is used to describe maintenance operations, such as cleanings and calibrations. An entry consists of the time and date, the type of maintenance, and a description.

PERFORMANCE DATE & TIME	date and time of last calibration
TYPE	maintenance type (CLEANING or CALIBRATION)
DESCRIPTION	maintenance description

EXPRESS Syntax:

```
ENTITY maintenance;
  performance_date_time : date_and_time;
  type                  : maintenance_type;
  description           : text;
UNIQUE
  performance_date_time;
END_ENTITY;
```

3.7 Physical Characteristic

The PHYSICAL CHARACTERISTIC type describes the physical limitations, the location, the geometry, and the devices comprising the SLM.

SAMPLE LIMITATIONS	lower and upper limits of samples the SLM is able to process properly
SLM LOCATION	location of the SLM within the workcell
SLM GEOMETRY	geometry of the SLM
OWNED SPACE GEOMETRY	geometry of the owned spaces
SLM DEVICES	list of devices making up the SLM

EXPRESS Syntax:

```
ENTITY physical_characteristic;
  sample_limitations   : OPTIONAL SET [1:?] OF range;
  slm_location         : OPTIONAL location;
  slm_geometry         : OPTIONAL geometry;
  owned_space_geometry : OPTIONAL geometry;
  slm_devices          : OPTIONAL SET [1:?] OF device;
END_ENTITY;
```

3.8 Port

The PORT type describes data and material ports of the SLM. Each port is described by an identifier, a description, a category, a port type, an access type, the current lock state, the port capacity, the current contents, the contents range, a location, a geometry, and a reference (pointer) to a list of port indices. Port indices are used to address sub-units of a port. For example, each position in a rack could be represented by a port index.

PORT ID	port identifier
DESCRIPTION	description of port
CATEGORY	port category, e.g., RACK, CONTAINER, etc.
TYPE	type of port (DATA or MATERIAL)
ACCESS TYPE	type of access (IN, OUT, or INOUT)
CURRENT LOCK STATE	current lock state of the port (LOCKED or UNLOCKED)
CAPACITY	capacity of port
CURRENT CONTENTS	current contents of port
CONTENTS RANGE	contents limitations, e.g., number of vials in an input or output rack
PORT LOCATION	physical location of the port, relative to the SLM location; address of data port
PORT GEOMETRY	description of port geometry
PORT INDICES	reference (pointer) to list of port indices

Each port index can be described with the same characteristics as the port itself. Therefore, the representation of a port index is identical to that of the port except for the port indices reference list.

PORT INDEX ID	identifier of port index
DESCRIPTION	description of port index
CATEGORY	port category, e.g., RACK, CONTAINER, etc.
TYPE	type of port (DATA or MATERIAL)
ACCESS TYPE	type of access (IN, OUT, or INOUT)
CURRENT LOCK STATE	current lock state of the port index (LOCKED or UNLOCKED)
CAPACITY	capacity of port index
CURRENT CONTENTS	current contents of port index
CONTENTS RANGE	physical contents limitations, e.g., number of vials in an input or output rack
INDEX LOCATION	physical location of port index, relative to port location; address of data port
INDEX GEOMETRY	description of port index geometry

EXPRESS Syntax:

```
ENTITY port;
  port_id            : identifier;
  description        : text;
  category           : label;
  type               : port_type;
  access_type        : port_access_type;
  current_lock_state : lock_type;
  capacity           : OPTIONAL measure;
  current_contents   : OPTIONAL measure;
  contents_range     : OPTIONAL SET [1:?] of range;
  port_location      : OPTIONAL location;
  port_geometry      : OPTIONAL geometry;
  port_indices       : OPTIONAL SET [1:?] OF port_index;
UNIQUE
  port_id;
END_ENTITY;


ENTITY port_index;
  port_index_id      : identifier;
  description        : text;
  category           : label;
  type               : port_type;
  access_type        : port_access_type;
  current_lock_state : lock_type;
  capacity           : OPTIONAL measure;
  current_contents   : OPTIONAL measure;
  contents_range     : OPTIONAL SET [1:?] of range;
  index_location     : OPTIONAL location;
  index_geometry     : OPTIONAL geometry;
UNIQUE
  port_index_id;
END_ENTITY;
```

3.9 Resource

The RESOURCE type describes the resources available. Each entry is described by an identifier, a description, a category, and a type, as well as information about shareability, available quantities, location, physical limitations, and the port indices of the access and maintenance ports.

RESOURCE ID	resource identifier
TYPE	type of resource LOCAL or PUBLIC
DESCRIPTION	description of the resource
CATEGORY	category to which the resource belongs
SHARE INFO	information about shareability of resource (SHAREABLE or NOT_SHAREABLE)
CAPACITY TYPE	capacity of resource (FINITE or INFINITE)
QUANTITY RANGE	lower and upper limit of resource quantity available
CURRENT QUANTITY	currently available quantity
RESOURCE LOCATION	physical location of the resource; relative to the SLM location
RESOURCE GEOMETRY	geometry of the resource
ACCESS PORTS	list of ports or port indices that allow access to the resource
MAINTENANCE PORTS	list of ports or port indices for the maintenance of the resource

EXPRESS Syntax:

```
ENTITY resource;
  resource_id          : identifier;
  type                 : resource_type;
  description          : text;
  category             : label;
  share_info           : resource_share_type;
  capacity_type        : capacity_type
  quantity_range       : OPTIONAL range;
  current_quantity     : OPTIONAL measure;
  resource_location    : OPTIONAL location;
  resource_geometry    : OPTIONAL geometry;
  access_ports         : OPTIONAL LIST [1:?] OF port_reference;
  maintenance_ports    : OPTIONAL LIST [1:?] OF port_reference;
UNIQUE
  resource_id;
END_ENTITY;
```

3.10 State

The STATE type is used to describe a processing state of the SLM. A state is defined by a unique state identifier, a state description, an optional list of entry and exit events, and an optional list of previous and next states.

STATE ID	state identifier
DESCRIPTION	state description
ENTRY EVENTS	list of messages that cause a transition to this state
EXIT EVENTS	list of messages that cause a transition out of this state
PREVIOUS STATES	list of states from which a transition to this state is possible
NEXT STATES	list of states that may follow this state

EXPRESS Syntax:

```
ENTITY state
  state_id          : identifier;
  description       : text;
  entry_events      : OPTIONAL SET [1..?] OF command_or_event;
  exit_events       : OPTIONAL SET [1..?] OF command_or_event;
  previous_states : OPTIONAL SET [1..?] OF state_reference;
  next_states       : OPTIONAL SET [1..?] OF state_reference;
UNIQUE
  state_id;
END_ENTITY;
```

3.11 System Variable

The SYSTEM VARIABLE type is used to describe SLM system variables that may have an influence on the operations of the SLM. They also may provide important information about the SLM to the TSC. Each entry is described by an identifier, a variable description, the data type of the variable, the access type and privilege, a variable category, an identifier of the device to which the variable belongs, a category, a data type, an initial value, a value range, and a current value.

VARIABLE ID	identifier of variable
DESCRIPTION	description of variable
DATA TYPE	data type of the variable
ACCESS TYPE	variable access type (READ_ONLY, WRITE_ONLY, or READ_WRITE)
ACCESS PRIVILEGE	access control
CATEGORY	variable category
DEVICE ID	identifier of device to which the variable belongs
INITIAL VALUE	initial value
VALUE RANGE	value range
CURRENT VALUE	current value

EXPRESS Syntax:

```
ENTITY system_variable;
  variable_id     : identifier;
  description     : text;
  data_type       : STRING;
  access_type     : variable_access_type;
  access_privilege: OPTIONAL label;
  category        : label;
  device_id       : OPTIONAL identifier;
  initial_value   : OPTIONAL STRING;
  value_range     : OPTIONAL SET [1:?] of range;
  current_value   : OPTIONAL STRING;
UNIQUE
  variable_id;
END_ENTITY;
```

Appendix

Appendix A. Device Capability Dataset in EXPRESS Syntax

The following chapter lists alphabetically the basic types and entities of Device Capability Datasets in the EXPRESS [9,10] syntax used in the previous chapter. The DCD components presented in Chapter 3 are not relisted in this chapter. However, they are components of the following EXPRESS schema.

```
(*-------------- SLM_CAPABILITY_DATASET_SCHEMA ------------*)

SCHEMA slm_capability_dataset_schema;

(*
  The following entities and types are used to describe
  the Device Capability Dataset of an SLM in EXPRESS syntax
  (ISO 10303-12)
*)


(*------------------ Schema References --------------------*)

(* References to ISO 10303-41 [14] *)
REFERENCE FROM support_resource_schema(identifier,
                                       label,
                                       text);

REFERENCE FROM date_time_schema(date_and_time,
                                coordinated_local_time);

REFERENCE FROM generic_measures_schema(unit,
                                       measure);


(* References to ISO 10303-42 [8] *)
REFERENCE FROM geometry_schema(placement);

REFERENCE FROM geometric_model_schema(shell_based_surface_model,
                                      face_based_surface_model,
                                      shell_based_wireframe_model,
                                      edge_based_wireframe_model,
                                      geometric_set,
                                      solid_model);


(*------------------ End of Schema References -------------*)
```

```
ENTITY argument;
  argument_id   : identifier;
  data_type     : STRING;
  default       : OPTIONAL STRING;
  value_range   : OPTIONAL SET [1:?] OF range;
END_ENTITY;


TYPE buffer_size = INTEGER;
WHERE
  WR1: { SELF >= 0 };
END_TYPE;


TYPE capacity_type = ENUMERATION OF (FINITE, INFINITE);
END_TYPE;


TYPE command_or_event = SELECT (command_reference,event_reference);
END_TYPE;


ENTITY command_reference;
  command_id      : identifier;
END_ENTITY;


ENTITY configuration_reference  SUBTYPE OF (command_reference);
END_ENTITY;


ENTITY data;
  data_id     : identifier;
  data_type   : STRING;
  description : OPTIONAL text;
  data_unit   : OPTIONAL unit;
END_ENTITY;


TYPE data_or_material = SELECT (data, material);
END_TYPE;


ENTITY device;
  device_id   : identifier;
  description : text;
END_ENTITY;
```

```
ENTITY event_reference;
  event_id     : identifier;
END_ENTITY;


TYPE interaction_type = ENUMERATION OF (REQUIRED_PRIMARY,
                                        REQUIRED_SECONDARY,
                                        OPTIONAL_PRIMARY,
                                        OPTIONAL_SECONDARY);
END_TYPE;


TYPE geometry = SELECT(shell_based_surface_model,
                       face_based_surface_model,
                       shell_based_wireframe_model,
                       edge_based_wireframe_model,
                       geometric_set,
                       solid_model);
END_TYPE;


ENTITY location  SUBTYPE (placement);
END_ENTITY;


TYPE lock_type = ENUMERATION OF (LOCKED, UNLOCKED);
END_TYPE;


TYPE maintenance_type = ENUMERATION OF (CLEANING, CALIBRATION);
END_TYPE;


ENTITY material;
  description : text;
  category    : label;
  quantity    : OPTIONAL measure;
END_ENTITY;


TYPE originator_type = ENUMERATION OF (TSC, SLM);
END_TYPE;


TYPE port_access_type = ENUMERATION OF (IN, OUT, INOUT);
END_TYPE;
```

```
ENTITY port_io;
  port : port_reference;
  what : data_or_material;
END_ENTITY;


TYPE port_type = ENUMERATION OF (DATA, MATERIAL);
END_TYPE;

ENTITY port_reference;
  port_id    : identifier;
  port_index : OPTIONAL identifier;
END_ENTITY;


TYPE priority_level = INTEGER;
WHERE
  WR1: { 1 <= SELF <= 255 };  -- (1 = highest, 255 = lowest)
END_TYPE;


ENTITY range;
  lower_limit : OPTIONAL measure;
  upper_limit : OPTIONAL measure;
END_ENTITY;


ENTITY request_reply
  request    : command_or_event;
  replies    : SET OF [1..?] of command_or_event;
  originator : originator_type;
  pre_states : OPTIONAL SET OF [1..?] of state_reference;
  post_state : OPTIONAL state_reference;
END_ENTITY;


ENTITY resource_reference;
  resource_id       : identifier;
  quantity          : OPTIONAL range;
  duration          : OPTIONAL coordinated_local_time;
  expected_at_port : OPTIONAL SET [1:?] OF port_reference;
END_ENTITY;

ENTITY required_resource_reference  SUBTYPE OF (resource_reference);
END_ENTITY;



TYPE resource_type = ENUMERATION OF (LOCAL, PUBLIC);
END_TYPE;
TYPE resource_share_type = ENUMERATION OF (SHAREABLE,
                             NOT_SHAREABLE);
END_TYPE;
```

```
ENTITY state_reference;
  state_id : identifier;
END_ENTITY;

TYPE variable_access_type = ENUMERATION OF (READ_ONLY,
                            WRITE_ONLY, READ_WRITE);
END_TYPE;


END_SCHEMA;

(*---------- End of SLM_CAPABILITY_DATASET_SCHEMA ---------*)
```

Appendix B. Clear Text Encoding

A Device Capability Dataset can be encoded in clear text format. The format used for the clear text encoding is defined by the ISO 10303-21 standard [14]. ISO 10303-21 requires that the DCD be represented in the EXPRESS language [9,10].

B.1 General structure

The Device Capability Dataset exchange structure can be stored in a sequential file organized in a modular manner. The structure consists of two sections: the HEADER and the DATA section. Each section begins with a section keyword that is immediately followed by a semicolon.

The HEADER section provides data relating to the structure of a Device Capability Dataset, while the DATA section provides the actual data to be transferred [14].

B.2 Example of a Device Capability Dataset

The following example uses the ISO 10303-21 clear text encoding standard to describe the capabilities of a portion of Zymark's Zymate System. This example represents only a limited number of capabilities and was not intended to cover all capabilities of a Zymate SLM.

```
(*
  -----------------------------------------------------------
  --  Zymate SLM Capability Dataset                        --
  --  in clear text encoded format (ISO 10303-21)          --
  -----------------------------------------------------------
*)

ISO-10303-21;

HEADER;

FILE_DESCRIPTION(('Capability Dataset file of Zymate SLM'),
                 '1');

FILE_NAME('STEP File for Zymate SLM',
          '1996-09-28 T18:45:00',
          ('Torsten Staab',
           'CAALS'),
          ('National Institute of Standards and Technology',
           'CSTL Div 839',
           'CAALS',
           'Gaithersburg',
           'USA'),
           'STEP VERSION 1.0',
           'Manually created',
           'Approved by ???');

FILE_SCHEMA(('slm_capability_dataset_schema'));

ENDSEC;
```

```
DATA;

(*---------------------- ADMINISTRATIVE ---------------------*)

#100 = ADMINISTRATIVE
       (
         (* manufacturer *)
         'Zymark',

         (* model number *)
         'Zymate',

         (* serial number *)
         '1234',

         (* software version number *)
         '2.10',

         (* dcd version number *)
         '1.3',

         (* update address *)
         'http://www.company.com/update',

         (* support address *)
         'http://www.company.com/support',

         (* information address *)
         'http://www.company.com/information',

         (* command buffer size *)
          0

       ); (* end of ADMINSTRATIVE *)

(*------------------------- ALARM ---------------------------*)

#200 = ALARM
       (
         (* alarm id *)
         'SYSTEM ERROR',

         (* description *)
         'issued if communication with the System V Controller
          fails, or if an EasyLab error is detected by the
          operating system'

         (* priority *)
         2,

         (*category *)
         'EasyLab VBX Event',

         (* alarm data format *)
         (
           ('message'      (* argument id *)
            'STRING',      (* data type *)
           $,              (* default *)
           $)              (* value range *)
          ),
```

```
         (* recovery commands *)
         $

      ); (* end of ALARM 'SYSTEM ERROR' *)


(*----------------------- COMMAND -------------------------*)

#300 = COMMAND
       (
         (* command id *)
         'ROBOT.EXERCISE',

         (* description *)
          'warms up the Zymate robot',

         (* category *)
         $,

         (* formal arguments *)
         $,

         (* duration *)
         ( 0,                    (* hour *)
           5,                    (* minute *)
           0.0,                  (* second *)
           (5, $, .behind.)      (* coordinated universal time offset *)
         ),

         (* measurement bounds *)
         $,

         (* required resources *)
         $,

         (* required configurations *)
         $,

         (* produced resources *)
         $,

         (* port inputs *)
         $,

         (* port outputs *)
         $,

         (* response data *)
         $

       ); (* end of COMMAND 'ROBOT.EXERCISE' *)

#301 = COMMAND
       (
         (* command id *)
         'PARK.HAND.A',

         (* description *)
          'robot parks hand at hand position A',

         (* category *)
         $,
```

```
          (* formal arguments *)
          $,

          (* duration *)
          $,

          (* measurement bounds *)
          $,

          (* required resources *)
          $,

          (* required configurations *)
          $,

          (* produced resources *)
          $,

          (* port inputs *)
          $,

          (* port outputs *)
          $,

          (* response data *)
          $

        ); (* end of COMMAND 'PARK.HAND.A' *)


#302 = COMMAND
        (
          (* command id *)
          'GET.HAND.A',

          (* description *)
           'robot gets general purpose hand A from park position A',

          (* category *)
          $,

          (* formal arguments *)
          $,

          (* duration *)
          $,

          (* measurement bounds *)
          $,

          (* required resources *)
          $,

          (* required configurations *)
          $,

          (* produced resources *)
          $,

          (* port inputs *)
          $,
```

```
    (* port outputs *)
    $,

    (* response data *)
    $

  ); (* end of COMMAND 'GET.HAND.A' *)


#303 = COMMAND
  (
    (* command id *)
    'GET.FROM.RACK',

    (* description *)
     'robot gets a vial from the rack at position n'

    (* category *)
    $,

    (* formal arguments *)
    ( 'n',                            (* argument id *)
      'INTEGER',                      (* data type *)
      $,                              (* default *)

      (* value range *)
      (
        (* lower limit *)
        (SPECIFIED_VALUE(1),
         UNITLESS(.no_units.)),
        (* upper limit *)
        (SPECIFIED_VALUE(4),
         UNITLESS(.no_units.))
      )
    ),

    (* duration *)
    $,

    (* measurement bounds *)
    $,

    (* required resources *)
    $,

    (* required configurations *)
    $,

    (* produced resources *)
    $,

    (* port inputs *)
    $,

    (* port outputs *)
    $,

    (* response data *)
    $

  ); (* end of COMMAND 'GET.FROM.RACK' *)
```

```
#304 = COMMAND
       (
         (* command id *)
         'PUT.INTO.RACK',

         (* description *)
          'robot puts a vial into the rack at position n'

         (* category *)
         $,

         (* formal arguments *)
         ( 'n',                              (* argument id *)
           'INTEGER',                        (* data type *)
           $,                                (* default *)

           (* value range *)
           (
             (* lower limit *)
             (SPECIFIED_VALUE(1),
              UNITLESS(.no_units.)),
             (* upper limit *)
             (SPECIFIED_VALUE(4),
              UNITLESS(.no_units.))
           )
         ),

         (* duration *)
         $,

         (* measurement bounds *)
         $,

         (* required resources *)
         $,

         (* required configurations *)
         $,

         (* produced resources *)
         $,

         (* port inputs *)
         $,

         (* port outputs *)
         $,

         (* response data *)
         $

       ); (* end of COMMAND 'PUT.INTO.RACK' *)

#305 = COMMAND
       (
         (* command id *)
         'GET.FROM.BALANCE',

         (* description *)
```

```
            'robot picks up an item from the balance',

           (* category *)
           $,

           (* formal arguments *)
           $,

           (* duration *)
           $,

           (* measurement bounds *)
           $,

           (* required resources *)
           $,

(* required configurations *)
           $,

           (* produced resources *)
           $,

           (* port inputs *)
           $,

           (* port outputs *)
           $,

           (* response data *)
           $

         ); (* end of COMMAND 'GET.FROM.BALANCE' *)


#306 = COMMAND
         (
           (* command id *)
           'PUT.INTO.BALANCE',

           (* description *)
            'robot puts the hold item into the balance',

           (* category *)
           $,

           (* formal arguments *)
           $,

           (* duration *)
           $,

           (* measurement bounds *)
           $,

           (* required resources *)
           $,

           (* required configurations *)
           $,

           (* produced resources *)
```

```
        $,

        (* port inputs *)
        $,

        (* port outputs *)
        $,

        (* response data *)
        $

      ); (* end of COMMAND 'PUT.INTO.BALANCE' *)


#307 = COMMAND
      (
        (* command id *)
        'OBTAIN.WEIGHT',

        (* description *)
         'balance weighs item and returns result',

        (* category *)
        $,

        (* formal arguments *)
        $,

        (* duration *)
        $,

        (* measurement bounds *)
        $,

        (* required resources *)
        $,

        (* required configurations *)
        $,

        (* produced resources *)
        $,

        (* port inputs *)
        $,

        (* port outputs *)
        $,

        (* response data *)
        (('WEIGHT',                                   (* data id *)
          'REAL',                                     (* data type *)
                                                      (* description *)
          'weight of item which is currently on balance',
          $)                                          (* data unit *)
         )

      ); (* end of COMMAND 'OBTAIN.WEIGHT' *)


#308 = COMMAND
      (
```

```
      (* command id *)
      'CAP',

      (* description *)
       'vial in the capping station will be capped',

      (* category *)
       $,

      (* formal arguments *)
      $,

      (* duration *)
      $,

      (* measurement bounds *)
      $,

      (* required resources *)
      $,

      (* required configurations *)
      $,

      (* produced resources *)
      $,

      (* port inputs *)
      $,

      (* port outputs *)
      $,

      (* response data *)
      $

    ); (* end of COMMAND 'CAP' *)


#309 = COMMAND
    (
      (* command id *)
      'UNCAP',

      (* description *)
       'vial in the capping station will be uncapped',

      (* category *)
       $,

      (* formal arguments *)
      $,

      (* duration *)
      $,

      (* measurement bounds *)
      $,

      (* required resources *)
      $,
```

```
          (* required configurations *)
          $,

          (* produced resources *)
          $,

          (* port inputs *)
          $,

          (* port outputs *)
          $,

          (* response data *)
          $

        ); (* end of COMMAND 'UNCAP' *)


#310 = COMMAND
        (
          (* command id *)
          'REMOTE_CTRL_GRANTED',

          (* description *)
           'TSC grants remote control to SLM',

          (* category *)
          $,

          (* formal arguments *)
          $,

          (* duration *)
          $,

          (* measurement bounds *)
          $,

          (* required resources *)
          $,

          (* required configurations *)
          $,

          (* produced resources *)
          $,

          (* port inputs *)
          $,

          (* port outputs *)
          $,

          (* response data *)
          $

        ); (* end of COMMAND 'REMOTE_CTRL_GRANTED' *)

#311 = COMMAND
        (
          (* command id *)
```

```
        'REMOTE_CTRL_DENIED',

        (* description *)
         'TSC denies the SLM's request for remote control mode',

        (* category *)
        $,

        (* formal arguments *)
        $,

        (* duration *)
        $,

        (* measurement bounds *)
        $,

        (* required resources *)
        $,

        (* required configurations *)
        $,

        (* produced resources *)
        $,

        (* port inputs *)
        $,

        (* port outputs *)
        $,

        (* response data *)
        $

      ); (* end of COMMAND 'REMOTE_CTRL_DENIED' *)
 (*------------------------- EVENT ---------------------------*)

#400 = EVENT
      (
        (* event id *)
        'COMMAND',

        (* description *)
        'issued for each command string sent to System V
         Controller',

        (* priority *)
        3,

        (*category *)
        'EasyLab VBX Event',

        (* event data format *)
        (
          ('message'   (* argument id *)
           'STRING',   (* data type *)
           $,          (* default *)
           $)          (* value range *)
         )

      ); (* end of EVENT 'COMMAND' *)
```

```
#401 = EVENT
        (
          (* event id *)
          'DONE',

          (* description *)
          'issued when System V Controller completes execution
           of a command and returns a Z> prompt',

          (* priority *)
          3,

          (*category *)
          'EasyLab VBX Event',

          (* event data format *)
          $,

        ); (* end of EVENT 'DONE' *)


#402 = EVENT
        (
          (* event id *)
          'PROMPT',

          (* description *)
          'issued when the System V Controller is waiting for
           user input',

          (* priority *)
          3,

          (*category *)
          'EasyLab VBX Event',

          (* event data format *)
          (
            ('message'    (* argument id *)
             'STRING',    (* data type *)
             $,           (* default *)
             $)           (* value range *)
           )

        ); (* end of EVENT 'PROMPT' *)


#403 = EVENT
        (
          (* event id *)
          'RESPONSE',

          (* description *)
          'issued for each line of text received from the System V
           Controller that is not an echo of a command or a
           prompt'

          (* priority *)
          3,
```

```
        (*category *)
        'EasyLab VBX Event',

        (* event data format *)
        (
          ('message'    (* argument id *)
           'STRING',    (* data type *)
           $,           (* default *)
           $)           (* value range *)
        )

      ); (* end of EVENT 'PROMPT' *)


#404 = EVENT
      (
        (* event id *)
        'REQUEST_REMOTE_CTRL',

        (* description *)
        'SLM request a transition from local to remote control',

        (* priority *)
        $,

        (*category *)
        $,

        (* event data format *)
        $
      ); (* end of EVENT 'REQUEST_REMOTE_CTRL' *)


(*---------------------- INTERACTION ------------------------*)

#500 = INTERACTION
      (
        (* interaction id *)
        'TRANSFER REMOTE CONTROL',

        (* description *)
        'This interaction is used by the equipment to request
         remote control mode',

        (*type *)
        .REQUIRED_SECONDARY.,

        (* transactions *)
        (
          (
            (* request *)
            'REQUEST_REMOTE_CTRL',

            (* replies *)
            (
              'REMOTE_CTRL_GRANTED',
              'REMOTE_CTRL_DENIED'
            ),

            (* originator *)
            .SLM.,
```

```
            (* pre states *)
            (
              'LOCAL'
            ),

            (* post states *)
            (
              'REMOTE',
              'LOCAL'
            )
          )
        )
      ); (* end of INTERACTION 'TRANSFER REMOTE CONTROL' *)


(*--------------------- RESOURCE ----------------------*)

(* no resource definition for this SLM *)


(*---------------------- MAINTENANCE ------------------------*)

(* no maintenance definition for this SLM *)


(*---------------- PHYSICAL CHARACTERISTIC ------------------*)

#600 = PHYSICAL_CHARACTERISTIC
       (
         (* sample limititations *)
         $,

         (* slm location *)
         $,

         (* slm geometry *)
         $,

         (* owned space geometry *)
         $,

         (* slm devices *)
         (
           (SYS_V_CTRL, 'System V Controller'),
           ('ROBOT',    'Zymate II Robot'),
           ('BAL',      'Mettler Balance'),
           ('CAP',      'Capping Station'),
           ('MLS',      'Master Laboratory Station'),
           ('PEC',      'Power and Event Controller')
         )

       ); (* end of PHYSICAL_CHARACTERISTIC *)


(*------------------------- PORT ---------------------------*)

#700 = PORT
       (
         (* port id *)
         'RACK 1',

         (* description *)
```

```
         'RACK with 50 positions',

         (* category *)
         'RACK',

         (* type *)
         .MATERIAL.,

         (* access type *)
         .INOUT.,

         (* current lock state *)
         .UNLOCKED.,

         (* capacity *)
         (SPECIFIED_VALUE(50),
          UNITLESS(.no_units.)),

         (* current contents *)
         $,

         (* contents range *)
         (
           (* lower limit *)
           (SPECIFIED_VALUE(1),
            UNITLESS(.no_units.)),
           (* upper limit *)
           (SPECIFIED_VALUE(50),
            UNITLESS(.no_units.))
         ),

         (* port location *)
         $,                    (* still needs to be defined! *)

         (* port geometry *)
         $,                    (* still needs to be defined! *)

         (* port indices *)
         $

       ); (* end of PORT 'RACK 1' *)


(*--------------------- PORT INDICES ------------------------*)

(* no port indices definition for this SLM *)


(*------------------------- STATE ---------------------------*)

#800 = STATE
       (
         (* state identifier *)
         'LOCAL',

         (* description *)
         'SLM is under local control',

         (* entry events *)
         (
           'REQUEST_LOCAL_CTRL'
         ),
```

```
        (* exit events *)
        (
          'OFFLINE'
        ),

        (* previous states *)
        {
          'REMOTE'
        ),

        (* next states *)
        (
          'REMOTE'
        )
      ); (* end of STATE 'LOCAL' *)


#801 = STATE
      (
        (* state identifier *)
        'REMOTE',

        (* description *)
        'SLM is under remote control',

        (* entry events *)
        (
          'REQUEST_REMOTE_CTRL'
        ),

        (* exit events *)
        (
          'REMOTE_CTRL_GRANTED'
        ),

        (* previous states *)
        {
          'LOCAL'
        ),

        (* next states *)
        (
          'LOCAL'
        )
      ); (* end of STATE 'REMOTE' *)



(*--------------------- SYSTEM VARIABLE ---------------------*)

#900 = SYSTEM_VARIABLE
      (
        (* variable id *)
        'SPEED',

        (* description *)
        'speed of robot movements',

        (* data type *)
        'REAL',
```

```
          (* access type *)
          .READ_WRITE.,

          (* category *)
          'Zymate II Core System',

          (* device id *)
          'ROBOT',

          (* initial value *)
          '1.0',

          (* value range *)
          (
            (* lower limit *)
            (SPECIFIED_VALUE(0.2),
             UNITLESS(.no_units.)),
            (* upper limit *)
            (SPECIFIED_VALUE(1.0),
             UNITLESS(.no_units.))
          ),

          (* current value *)
          $

        ); (* end of SYSTEM VARIABLE 'SPEED' *)


#901 = SYSTEM_VARIABLE
        (
          (* variable id *)
          'TARGET.WEIGHT',

          (* description *)
          'desired weight value of solid addition',

          (* data type *)
          'REAL',

          (* access type *)
          .READ_WRITE.,

          (* access privilege *)
          'ALL',

          (* category *)
          'Weighing',

          (* device id *)
          'BALANCE',

          (* initial value *)
          $,

          (* value range *)
          (
            (* lower limit (0.0001 g) *)
            (SPECIFIED_VALUE(100),
             NAMED_UNIT(.micro.,
                        .gram.,
                        (0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0))),
```

```
            (* upper limit (250 g) *)
            (SPECIFIED_VALUE(25),
             NAMED_UNIT(.deca.,
                        .gram.,
                        (0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)))
          ),

          (* current value *)
          $

        ); (* end of SYSTEM VARIABLE 'TARGET.WEIGHT' *)


#902 = SYSTEM_VARIABLE
        (
          (* variable id *)
          'WEIGHT.VALUE',

          (* description *)
          'value is acquired by OBTAIN WEIGHT, while container
           still on balance',

          (* data type *)
          'REAL',

          (* access type *)
          .READ_WRITE.,

          (* category *)
          'Weighing',

          (* device id *)
          'BALANCE',

          (* initial value *)
          $,

          (* value range *)
          (
            (* lower limit (0.0001 g) *)
            (SPECIFIED_VALUE(100),
             NAMED_UNIT(.micro.,
                        .gram.,
                        (0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0))),
            (* upper limit (250 g) *)
            (SPECIFIED_VALUE(25),
             NAMED_UNIT(.deca.,
                        .gram.,
                        (0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)))
          ),

          (* current value *)
          $

        ); (* end of SYSTEM VARIABLE 'WEIGHT.VALUE' *)


ENDSEC;

END-ISO-10303-21;
```

References

[1] Salit, Marc L.; Guenther, Franklin R.; Kramer, Gary W.; and Griesmeyer, J. Michael, *Analytical Chemistry*, **66,** 361A-367A (1994).

[2] Kingston, Howard M. (Skip), *Analytical Chemistry*, **61,** 1381A-1384A (1989).

[3] Kramer, Gary W., ed., Communications Protocol Specification, CAALS-I Communication, National Institute of Standards and Technology, CAALS internal document, 134pp, 1996.

[4] Staab, Torsten A.; Bernhöft, Uwe; and Kramer, Gary W., CAALS High Level Communication Protocol (HLCP), National Institute of Standards and Technology, CAALS internal document, 39pp, 1996.

[5] Staab, Torsten A.; Grandsard, Peter J.; and Kramer, Gary W., CAALS Common Commands, National Institute of Standards and Technology, CAALS internal document, 76pp, 1995.

[6] Salit, Marc L. and Griesmeyer, J. Michael, *Laboratory Robotics and Automation*, **9,** 113-118 (1997).

[7] Griesmeyer, J. Michael, General Equipment Interface Definition, Sandia National Laboratories, 54pp, 1994.

[8] *PostScript Printer Description File Format Specification*, Vers. 4.2, Adobe Systems, Inc., 184pp, March 29, 1994, available via ftp://ftp.adobe.com/pub/adobe/devrelations/devtechnotes/pdffiles/5003_43.ppdspec.pdf (last visited 03/99).

[9]
a. ISO 10303-11:1994; Industrial automation systems and integration–Product data representation and exchange–*STEP Part 11: Description methods: The EXPRESS language reference manual*, 208pp, 1994.
b. ISO/TR 10303-12:1997; Industrial automation systems and integration–Product data representation and exchange–*STEP Part 12: Description methods: The EXPRESS-I language reference manual*, 111pp, 1997.

[10] Schenck, Douglas A. and Wilson, Peter R., *Information Modeling: The EXPRESS Way*, Oxford University Press, New York, 388pp, 1994.

[11] ISO 10303-1:1994; Industrial automation systems and integration–Product data representation and exchange–*STEP Part 1: Overview and fundamental principles*, 17pp, 1994.

[12] ISO 10303-42:1994; Industrial automation systems and integration--Product data representation and exchange–*STEP Part 42: Integrated generic resources: Geometric and topological representation*, 241pp, 1994.

[13] Day, Anthony J., Capability Dataset Structure for a Laboratory Information Management System, National Institute of Standards and Technology, CAALS internal document, 34pp, 1995.

[14] ISO 10303-41:1994; Industrial automation systems and integration--Product data representation and exchange–*STEP Part 41: Integrated generic resources: Fundamentals of product description and support*, 211pp, 1994.

[15] ISO 10303-21:1994; Industrial automation systems and integration--Product data representation and exchange–*STEP Part 21: Implementation methods: Clear text encoding of the exchange structure*, 57pp, 1994.